Forth – die ewig junge Programmiersprache

Von Carsten Strotmann (veröffentlicht am 25.12.2020 auf Heise-Online)

Forth ist so ungewöhnlich, dass Entwickler beim Versuch, ihr Wissen über Programmiersprachen auf Forth anzuwenden, sich oft irritiert abwenden. Das ist schade.

(Bild: kentoh / Shutterstock.com)

Forth ist eine "Concatenative"-Programmiersprache. Das heißt, es gibt nur minimale Syntaxregeln, und Befehle beziehungsweise Funktionen (im Forth-Sprachgebrauch "Wörter") werden mit Leerzeichen getrennt hintereinander geschrieben. Alle Bestandteile, die in populären Programmiersprachen durch Syntax umgesetzt sind (Kommentare, Variablendefinitionen, Kontrollstrukturen wie Schleifen oder Entscheidungen mit IF-THEN-ELSE) sind unter Forth ganz normale "Wörter".

Im Umkehrschluss bedeutet das, dass Programmierer "ihr" Forth selbst in beliebiger Weise anpassen können. Forth ist wie ein Baukasten für eigene Programmiersprachen, die es Programmierern und Programmiererinnen erlauben, möglichst elegant die gewünschte Anwendung zu implementieren.

EVO PDF Tools Demo

Forth zum Anschauen

Der Forth-Quellcode ist gewöhnungsbedürftig. Die Struktur wird nicht von der Sprache vorgegeben, es ist Programmierern überlassen, den Quellcode lesbar zu formatieren. Da in den ersten Jahren von Forth auch der Platz für Quellcode auf Speichermedien stark begrenzt war, hatte sich Forth leider den Ruf einer Write-only-Programmiersprache zugezogen. Es braucht ein wenig Disziplin, um schön lesbare Forth-Programme zu schreiben.

Um ein Gefühl für Forth zu bekommen, wird im nachfolgenden Auschnitt ein Teil des Forth-Quellcodes aus dem "Tetris for terminals" des GNU/Forth (Datei *tt.fs*) näher erklärt. Der Code implementiert einen Pseudo-Zufallszahlengenerator (Pseudo-Random-Number-Generator oder PRNG), der im Tetris-Spiel zum Einsatz kommt. Im Quellcode werden bestehende Forth-Wörter wie `dup`, `+` und `!` zu neuen Wörtern (Funktionen, Unterroutinen) zusammengesetzt:

```
\ stupid random number generator

variable seed

: randomize     time&date    \ Stack: sekunde minute stunde tag monat jahr
                + + + + +    \ addiert sekunde, minute, stunde, tag, monat und jahr
                seed ! ;     \ Ergebis in seed speichern

$10450405 constant generator

: rnd  ( -- n )  seed @ generator um* drop 1+ dup seed ! ;
```

Die erste Zeile ist ein Kommentar, eingeleitet mit dem Forth-Wort `\`. Dabei ist `\` keine Syntax, wie man es von anderen Programmiersprachen kennt, sondern ein Forth-Wort (eine Funktion), die beim Lesen des Quelltexts direkt ausgeführt wird. Diese Funktion liest alle Zeichen im Quelltext bis zum Ende der Zeile und verwirft diese Zeichen. Damit ist der Kommentartext für das Forth-System nicht mehr sichtbar, und die Funktion eines Kommentarzeichens ist implementiert. Haben Programmierer eigene Vorstellungen, welches Zeichen einen Kommentar einleiten sollte, fügen sie einfach dem Forth-System ein eigenes, neues Kommentarwort hinzu.

Die Zeile `variable seed` ist einfach zu verstehen. Sie erzeugt eine Variable. Eine Forth-Variable kann einen numerischen Wert speichern, welcher der nativen Datenbus-Größe der CPU entspricht. Also 16, 32, 64 oder im Fall von RISC-V auch schon 128 Bit. Dabei ist das Forth-Wort `variable` keine spezielle Syntax, sondern nur eine in Forth geschriebene Funktion, die Speicher reserviert und mit dem angegebenen Namen verbindet. Der Name der Variable, hier `seed`, ist danach ein neues Forth-Wort, das bei der Ausführung die Adresse der Variable auf den Datenstapel (Data-Stack) legt (hierzu später mehr). Diese Variable ist bei den meisten Forth-Systemen schon die einzige Datenstruktur, die das System mitliefert. Arrays, Structs, Objekte und andere Datenstrukturen erzeugen Forth-Programmierer über eigene, individuelle Forth-Wörter. Das Erzeugen eigener Datenstrukturen in Forth ist nicht schwer, und auch bei populären Programmiersprachen sind komplexere Datenstrukturen oft selbst zu erstellen. Also warum nicht gleich von Anfang an.

Das Forth-Wort `:` (Colon) aktiviert den Forth-Compiler. Alle nachfolgenden Wörter, bis zum abschließenden Forth-Wort `;` (Semis), werden nicht direkt ausgeführt, sondern in ein neues Forth-Wort einkompiliert. Der Name (oder Bezeichner) des neuen Worts folgt direkt auf `:`.

Dieses neue Forth-Wort hat die Aufgabe, den Zufallszahlengenerator für das Tetris-Spiel zu initialisieren (der hier gezeigte Zufallszahlengenerator ist sehr schlicht, reicht für Spiele, aber sollte nicht für sicherheitskritische Anwendungen benutzt werden).

Das Forth-Wort `time&date` legt sechs Werte (Sekunde, Minute, Stunde, Tag, Monat, Jahr) auf den Stack des Forth-Systems ab. In einem Forth-System werden alle Ein- und Ausgabeparameter für Funktionen über einen Daten-Stack implizit übergeben. "Implizit" heißt hier, dass die Anzahl der Parameter bei der Definition des Forth-Worts, und auch beim Aufruf, nicht angegeben werden. Der Stack ist eine eingebaute LIFO-Datenstruktur (Last-In – First-Out) eines jeden Forth-Systems. Es erfordert ein wenig Übung, die Bewegung der Parameter auf dem Datenstapel mitzuverfolgen. Anfänger können sich mit ausführlichen Kommentaren helfen. Die Anzahl der Parameter ist im Quelltext nicht erkennbar, kann aber als Kommentar angegenben werden.

```
: randomize
    time&date     \ Stack: sekunde minute stunde tag monat jahr
    + + + + +     \ addiert sekunde, minute, stunde, tag, monat und jahr
    seed !        \ Ergebis in seed speichern
;
```

Die nachfolgenden Additionszeichen `+` sind Forth-Wörter, deren Funktion es ist, die jeweils oberen zwei Elemente auf dem Stack miteinander zu addieren und das Ergebnis auf den Stapel zurückzulegen. Forth bedient sich bei mathematischen Funktionen der Postfix-Schreibweise. Dabei werden erst die Werte auf den Stack gelegt und danach die mathematische Funktion aufgerufen, welche die Werte von Stack "konsumiert" und die Ergebnisse wieder auf dem Stack hinterlässt.

Hier wird aus dem aktuellen Datum und der aktuellen Uhrzeit Uhrzeit ein Startwert (Seed) für den Zufallszahlengenerator ermittelt. Das Wort zur Variablen `seed` legt die eigene Speicheradresse auf den Stack, und das nachfolgende Wort `!`, ausgesprochen "Store", speichert den errechneten Wert in die Speicherstelle der Variable `seed`. Dabei ist anzumerken, das `!` nicht nur in Variablen schreiben kann, sondern über entsprechende Eingabewerte auf dem Stack in beliebige Speicherbereiche schreibt.

Die folgende Zeile ist wieder einfacher zu verstehen:

```
$10450405 constant generator
```

Sie erzeugt ein neues Forth-Wort mit dem Namen `generator`. Es ist eine numerische Konstante, die mit dem Wert Hexadezimal 10450405 belegt ist. Wird dieses neue Forth-Wort im weiteren Programm benutzt, legt es diesen Wert auf den Stack.

In der letzten Zeile des kleinen Beispiels wird nun ein Forth-Wort erzeugt, das bei der Ausführung eine neue Zufallszahl erzeugt und diese auf dem Stack hinterlässt:

EVO PDF Tools Demo

```
: rnd   ( -- n )            \ erzeugt Zufallszahl
  seed @                    \ lade aktuelle Saat des PRNG
  generator um* drop 1+     \ errechne Zufallszahl
  dup                       \ Zahl verdoppeln
  seed !                    \ und eine Kopie als neue Saat speichern
;
```

Colon (`:`) schaltet wieder den Compiler an, und `rnd` ist der Name des neuen hier definierten Forth-Worts. Die Klammern `(` und `)` stellen eine weitere Form eines Kommentars dar (wobei `(` ein Forth-Wort ist, das alle Zeichen im Quelltext bis zum Zeichen `)` konsumiert, verwirft und damit dem Forth-System vorenthält). Das ist ein Stack-Kommentar, der als Dokumentation für Programmierer dient und beschreibt, welche Eingabe- und Ausgabeparameter diese Funktion erwartet beziehungsweise erzeugt. In dem Fall wird eine natürliche (Zufalls-)Zahl `n` nach dem Aufruf von `rnd` auf dem Stack hinterlassen.

Ab `seed` beginnt der Körper der neuen Funktion. `seed` als Variable von oben legt die eigene Speicheradresse auf den Stack. Das Forth-Wort `@` (gesprochen "fetch") ist das Gegenstück zu `!` (store) und konsumiert die Adresse als obersten Eintrag auf dem Stapel und legt als Ergebnis den an der Adresse im Hauptspeicher gespeicherten Wert zurück auf den Stack.

Das Forth-Wort `generator` (eine numerische Konstante) legt den Wert Hexadezimal 10450405 auf den Stack, welches das Wort `um*` (unsigned double multiply "u-m-star") mit dem Wert aus `seed` multipliziert und als doppelt-genaue Zahl auf dem Stack hinterlässt.

Das Wort `drop` verwirft den oberen Wert des Stacks, hier den höherwertigen Teil der doppelt-genauen Zahl (der Zufall spielt sich im unteren Bereich der Zahl ab). Dieses Ergebnis, der untere Teil der doppelt-genauen Zahl, wird mit dem Wort `1+` um eins erhöht und dann mit dem Wort `dup` dupliziert. `dup` liest den obersten Eintrag auf dem Datenstapel und legt eine Kopie davon auf ebenjenen. Die Kopie des Werts wird nun als neue Saat in die Variable `seed` gespeichert, während der originale Wert auf dem Stack verbleibt und als Ergebnis der Funktion auf weitere Verwendung im Programm wartet. Das Wort `;` (Semicolon) schließt die Definition des neuen Forth-Worts `rnd` ab, schaltet den Forth-Compiler aus und übergibt die Kontrolle wieder an das interaktive Forth-System.

Warum Forth?

Fragt man Forth-Anwender nach den Vorteilen der Sprache, bekommt man unterschiedliche Antworten. Da ist einmal der geringe Speicherverbrauch eines Forth-Systems: auf Embedded-Systemen mit ARM- oder RISV-V-CPU verbraucht ein vollständiges Forth-System oft weniger als 16 KiB RAM- oder Flash-Speicher (ja, Kilobyte). Auf kleinen Systemen bleibt mehr Platz für die Anwendung. Bei der Produktion von Hardware kann das eine wichtige Kostenersparnis sein. Mit dem geringen Speicherverbrauch kommt eine hohe Energieeffizienz, wichtig für Anwendungen, die über Jahre batteriegespeist laufen müssen (z. B. Tiefseesensoren) oder nur im geringen Maße selbst Strom erzeugen können (Weltraummissionen wie der Philae Lander der Rosetta-Mission).

Intern ist Forth als Programmiersprache einfach aufgebaut – so einfach, dass Programmierer nach einigen Jahren Forth-Programmierung anfangen, ein eigenes Forth zu schreiben. Durch ihre Einfachheit ist die Sprache auf vielen Systemen (CPU-Architekturen, Mikrocontroller-Boards und Betriebssystemen) zu finden. Vermutlich keine andere Programmiersprache wurde auf so viele Rechnersysteme portiert.

Das gibt Entwicklern eine enorme Flexibilität: Die Hardware oder das Betriebssystem müssen nicht schon zu Anfang eines Projekts fest geschrieben werden, sondern lassen sich auch später noch ändern, ohne dass schon erstellter Programmcode dabei überflüssig würde. Und findet man ein Rechnersystem, auf dem es noch kein Forth gibt, dann schreibt man selbst eines (oder lässt es schreiben). Eine Portierung von Forth auf eine neue CPU-Architektur oder ein neues Mikrocontroller-Board ist in der Regel in zwei bis vier Tagen abgeschlossen. Damit wird das Forth-Projekt beliebig skalierbar, vom Mikrocontroller mit 16 KByte Flash und 0,5 KByte RAM bis zum Multi-CPU-Server oder Supercomputer lässt sich der gleiche Quellcode ausführen.

Hardwarenahe Entwickler schätzen das vorhersagbare Laufzeitverhalten von Forth. Die Sprache direkt auf Hardware (ohne Betriebssystem) ausgeführt, erlaubt harte Echtzeitprogrammierung, bei der die Ablauf- und Antwortgeschwindigkeit der Software vollständig berechenbar und prüfbar ist.

Forth-Systeme sind interaktiv. Das heißt, sie haben eine eigene Kommandozeilenschnittstelle eingebaut, auch bei Embedded-Systemen. Entwickler können neue Ideen schnell interaktiv in Forth direkt auf der Zielhardware ausprobieren, ohne einen langsamen "Edit-compile-run"-Zyklus, aus dem bei Embedded-Systemen oft ohne Forth ein "Edit-compile-flash-reset-dead"-Zyklus wird.

Da diese interaktive Kommandozeilenschnittstelle und der Forth-Compiler platzsparend implementiert sind und nicht viel Flash- oder RAM-Speicher verbrauchen, bleiben sie oft in der fertigen Anwendung (meist "versteckt" und per Sonderkommando, Interrupt oder Tastenkombination aktivierbar). Wartungstechniker spielen über diese Schnittstelle bei bestehenden Systemen "im Feld" Änderungen ein oder testen Ad-hoc-Patches aus. Forth selbst ist die Entwicklungsumgebung, oft benötigt man nur einen Laptop als "Terminal" über USB, TCP/IP oder serielle Leitung.

Das "Killer-Feature" von Forth ist jedoch die Erweiterbarkeit des Systems. Hat man sich mit ihm angefreundet, muss man es auch nicht verlassen, wenn etablierte oder moderne Konzepte wie objektorientierte, funktionale, aspektorientierte, logische oder deklarative Programmierung benötigt werden: Diese Konzepte lassen sich mit moderatem Aufwand in jedes Forth-System einbauen. Somit wächst Forth mit den Anforderungen der Programmierer und den neuen Erkenntnissen aus der Informatik – sozusagen als ewig junge Programmiersprache.

Modernes Forth

Verglichen mit anderen, populären Programmiersprachen ist Forth recht alt, es wurde 1968 von *Charles "Chuck" Moore* entwickelt. Dabei war Forth immer als Open-Source-Software verfügbar – auch wenn es den Begriff damals noch nicht gab –, die Forth-Implementierungen der *Forth Interest Group (FIG)* Ende der 1970er-Jahre sorgten für einen Popularitätsschub in der Zeit der Heimcomputer.

Über die Jahrzehnte wurden verschiedene Forth-Standards verabschiedet: *Forth-79*, *Forth-83*, *ANSI/ISO Forth 1994*. Den aktuellen Standard hat das Standard-Team, das sich aus Vertretern der Forth-Entwicklergemeinde und kommerziellen Forth-Anbietern zusammensetzt, 2012 veröffentlicht. Traditionell trifft es sich einmal im Jahr im Rahmen der *Euro-Forth-Konferenz*. Die Entwicklung des Standards ist nicht abgeschlossen, für die nächsten Jahre wird ein neuer Standard erwartet. Dabei ist er nur eine Empfehlung an die Entwickler, um portable Forth-Programme schreiben zu können. Viele Forth-Systeme bieten Funktionen, die weit über den Standard hinaus gehen.

EVO PDF Tools Demo

GNU/Forth (auch *gforth* genannt) ist die populärste Forth-Implementierung für PC-Systeme und steht unter der GPL. GNU/Forth ist in den Repositories der meisten *Linux*- und *BSD-Systeme* verfügbar und lässt sich auch mit *Windows*, *macOS* oder exotischen Systemen (*Haiku*, *MorphOS*) betreiben. GNU/Forth ist so etwas wie die inoffizielle Referenzimplementierung des Forth-Standards. Viele neue Ideen, die heute im Standard beschrieben sind, wurden unter GNU/Forth implementiert und getestet. Nach über 30 Jahren Entwicklung an GNU/Forth ist in den kommenden Monaten die Version 1.0 zu erwarten.

Forth in Embedded-Systemen

Auch bei den hardwarenahen Forth-Systemen gibt es interessante Entwicklungen: Die speziell bei Hobby-Elektronikern populäre Propeller-CPU-Architektur hat erst kürzlich mit der Propeller 2 CPU einen Nachfolger erhalten. Das Besondere: Diese CPU hat ein Forth direkt eingebaut (in einem kleinen ROM). Damit lässt sich die CPU allein in Betrieb nehmen, und Entwickler können direkt über Forth mit der CPU interagieren und die eigenen Elektronikprojekte testen. Der Propeller 2 lässt sich auch mit anderen Programmiersprachen wie C, Basic oder Assembler programmieren, jedoch ist Forth aufgrund der geringen Speicheranforderungen direkt eingebaut und immer verfügbar. Wie bei den frühen Homecomputern der 80er-Jahre, bei denen immer ein Basic nach dem Einschalten zum Programmieren einlud.

Wer auf Geschwindigkeit Wert legt, sollte sich *Mecrisp Forth* von *Matthias Koch* näher anschauen. Es übersetzt den Quellcode mit einem optimierenden Compiler direkt in die Maschinensprache des Zielsystems (und dabei läuft der Compiler auf dem Zielsystem, nicht auf einem PC). Neben ARM-Architekturen, MSP430 und RISC-V unterstützt Mecrisp-ICE die J1a CPU, eine Open-Source-CPU-Architektur für FPGA-Systeme. Diese Forth-CPU lässt sich mit komplett freier Software in FPGA-Systeme einspielen.

Für Freunde des Arduino (und generell fast aller 8-Bit ATMEL CPUs) gibt es *AmForth*. Aus Gründen der Speicherplatzeffizienz erzeugt es auf den Zielsystemen einen gefädelten Code (Threaded-Code), der weniger Geschwindigkeit, dafür aber mehr Programm im Flash-Speicher erlaubt. Neben den Atmel CPUs unterstützt AmForth MSP430-Systeme und neuerdings auch ARM (32-Bit) und RISC-V.

Sowohl AmForth als auch Mecrisp implementieren den aktuellen Forth-Standard und sind unter Open-Source-Lizenz verfügbar. Entwickler, die auf kommerziellen Support bei Forth-Projekten angewiesen sind, finden mit *SwiftForth (Forth Inc.; USA)* und *VFX-Forth (MPE; UK und Österreich)* ausgereifte Forth-Systeme für PCs und viele Embedded-CPU-Architekturen und Anbindung an zeitgemäße Frameworks wie *GTK+* unter Windows/Linux oder *Cocoa* unter macOS beziehungsweise an TCP/IP-Stacks, Audio-Frameworks der Betriebssysteme.

Fazit und Ausblick

Seit über 35 Jahren kümmert sich die *Forth-Gesellschaft e.V.* um die Forth-Programmierer im deutschsprachigen Raum. Auf der Webseite des Vereins finden sich Forth-Kurse, Videos von Konferenzen und die PDF-Ausgaben der Vereinszeitschrift *„Vierte Dimension“* zum kostenlosen Download. Die regelmäßigen regionalen Treffen von Forth-Freunden wurden im Pandemiejahr 2020 zu globalen Online-Ausgaben.

Bei GitHub finden sich nicht nur viele Forth-Systeme im Quelltext, mit dem Forth-Hub haben sich dort Forth-Programmierer zusammengefunden, um über Implementierungen und Programmierfragen zu diskutieren. Auf Exercism.io entsteht derzeit ein Forth-Kurs mit praktischen Übungen, bei dem Interessierte mithilfe von Mentoren die ersten Schritte in die Forth-Welt meistern können.

Forth ist keine Programmiersprache für jeden Programmierer oder alle Programmieraufgaben, jedoch öffnet die Beschäftigung mit Forth die Tür in eine andere, neue Welt mit vielen ungewöhnlichen Ideen, die sich in jeder Programmiersprache anwenden lassen. Und auch wenn man Forth nicht gleich im nächsten Projekt einsetzen möchte, die Forth-Konzepte helfen bei der Arbeit mit C/C++, Rust, Go, Java oder Python. Jeder Programmierer sollte einmal einen Blick über den Tellerrand wagen – auf Lisp, Smalltalk und eben auf Forth.

Carsten Strotmann

> *ist freiberuflicher Berater und Trainer zu den Themen DNS, DHCP, IPv6, Linux- und BSD-Betriebsysteme. Er wurde vor 30 Jahren mit dem Forth-Virus infiziert und freut sich, Forth weiterhin in spannenden Projekten einsetzen zu können. Neben Forth programmiert er unter anderem in C, Go, Ruby, Python, Scheme/Lisp, Nim und Zig.*

(ane)

Nachtrag eines Benutzers aus dem Heise-Forum:

Re: Der Artikel zeigt sehr eindrucksvoll, ...

> faun-dy schrieb am 25.12.2020 14:01:
>
> > Wenn der Autor nun auf Embedded kommt, da grausts mir erst recht. Schon in C ist die fehlerhafte Vewendung von Pointern eine der verbreitesten Fehlerklassen (die andere sind Konvertierungen und Überläufe). Aber einfach eine anonyme Adresse auf den Stack zu legen und dann "irgenwie" "irgendwas" zusuweisen ist mehr als sportlich. Im Embedded-Umfeld, wo physikalische Objekte damit gesteuert werden kann sowas sehr schnell als fahrlässig oder sogar sträflich angesehen werden. Insbesondere wenn damit Sach oder sogar Personenschäden einhergehen.
>
> Genau, da hast du den Artikel eben nicht gelesen, nicht verstanden, oder Carsten hat diesen Teil nicht ausführlich genug beschrieben. Der wesentliche Unterschied zu C ist eben, dass du eine Kommando-Zeile hast, an der du solche Sachen mal eben ausprobieren kannst. Und wenn sie funktionieren, dann baust du sie in deinen Quellcode ein.

EVO PDF Tools Demo

> Wenn ich in C programmiere (oder einer beliebigen anderen Edit-Compile-Link-Go-Sprache), dann fühle ich mich wie knietief durch Matsch watend programmieren. Alles, was ich so schreibe, muss dann im Kontext des Systems funktionieren, weil einfach so mal schnell ausprobieren ist nicht. Embedded erst recht nicht, weil da dann noch das Flashen hinzukommt, und das Debuggen mit den C-Werkzeugen deutlich schwieriger ist.
>
> Wenn man's nicht anders kennt, hofft man, dass einem der Compiler noch mehr Verifikation liefert oder so. Aber der Compiler hat an der Stelle oft kein Expertenwissen. Der kann vielleicht wissen, dass eine IO-Variable ein Byte oder ein Wort ist, aber nicht, wie jetzt genau man die Waschmaschine durch Schreiben auf dieses Byte dazu bringt, das Wasser einlaufen zu lassen. Zumal nicht mal die Tool-Chain das so genau weiß, weil das ja nur ein externes Bauteil an einem IO-Port dran ist.
>
> Programmieren von embedded-Systemen ist alles andere als formales Zeugs; man steckt da auch mal knietief in irgendeiner Art Schlamm (Nick Nelson berichtet auf der EuroForth öfter von Häufen von Tütensuppen oder einem meterhohen Schokoberg aus seiner Zeit als Automatisierer von Lebensmittelverarbeitung). Deshalb funktionieren dann am Ende die in Forth geschriebenen Systeme, während die in Ada geschriebenen zum Crash der Rakete führen — weil man glaubte, sie nicht ausprobieren zu müssen. Hat ja compiliert, und das war schwer genug, so oft wie der Compiler gemeckert hat!
>
> Unit-Tests und so kann man in Forth übrigens einfach so direkt in den Quelltext einbauen, dazu nutzt man dann eben den Interpreter, der ja immer da ist.